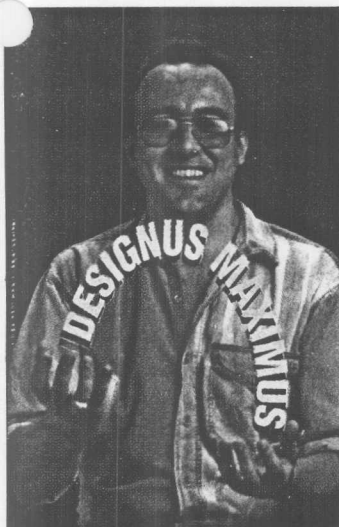


THE OUROBOROS

OF THE DIGITAL CONSCIOUSNESS: LINEAR-FEEDBACK-SHIFT REGISTERS



CLIVE "MAX" MAXFIELD,
INTERGRAPH ELECTRONICS

A variety of cultures worldwide has employed the Ouroboros—a symbol of a serpent or dragon devouring its own tail and thereby forming a circle to depict eternity or renewal. In the rarified realm of digital designers, an equivalent to the Ouroboros is the linear-feedback-shift register (LFSR), in which the output from a standard-shift register is manipulated and fed back to its input so as to cause the function to endlessly

cycle through a sequence of patterns.

In much the same way that the Ouroboros symbolizes renewal, the subject of LFSRs keeps on rearing its head. Strangely enough, although LFSRs are simple to construct and are useful for a wide variety of applications, designers often neglect them. You can construct one of the most common forms of an LFSR using a simple shift register with feedback from one or more points, known as taps, in the register chain (Figure 1). All of the registers share a common clock, which is omitted from the symbol representation in Figure 1. Unless stated otherwise, for the purpose of these discussions, the left bit is the LSB, and the right bit is the

The linear feedback shift register is a simple circuit that can be used in a variety of ways. You can create a maximal length pseudo-random sequence of values by tapping into the register at prescribed tap points to provide feedback to the register's input.

MSB. The taps for Figure 1 are at bit 0 and bit 2, referenced as [0,2].

You can generate data inputs for a LSFR by using an XOR or XNOR gate on the tap bits, and the remaining bits function as a standard shift register. The reason that LSFRs are so named is due to their XOR or XNOR feedback functions, which are linear operators. Both feedback function and tap selection determine the sequence of states generated by the LFSR (Figure 2). In this example, both LFSRs begin with the same initial state, but, due to different taps, their sequences rapidly diverge as clock pulses are applied. Following the second clock, the LFSR that has taps at [1,2] enters a loop comprising only two states (the actual loop sequence depends on the initial value).

In comparison, the LFSR with taps at [0,2] is maximal-length, because it sequences through every possible state (excluding the all-zero state) before returning to its initial value. A binary field with n bits can assume 2^n unique values, but an unmodified maximal-length LFSR with n regis-

LINEAR-FEEDBACK-SHIFT REGISTERS

ter bits sequences only through (2^n-1) states. LFSRs with XOR feedback paths don't sequence through the state where all bits are zero, and their XNOR counterparts don't sequence through the state where all of the bits are one (Figure 3). In addition, you can't permit an unmodified LFSR to enter its prohibited state during initialization on power-up, because there is no way for the LFSR to exit that state without external assistance from additional circuitry.

Each LFSR has a multitude of tap combinations that generate maximal-length sequences; the problem is weeding out the combinations that do from those that don't. Another problem is the fact that documented tap combinations often contain errors. To overcome this, I wrote a simple program that ascertains valid tap combinations. Using the program, I extracted all of the tap combinations for maximal-length LFSRs that have 2 to 32 bits. The program ran for several weeks and generated large quantities of

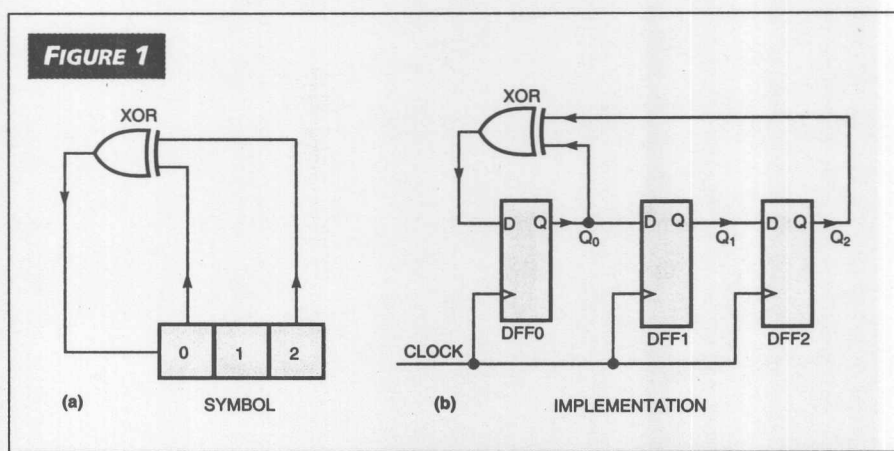


FIGURE 1
A symbolic (a) and an actual (b) implementation of an LFSR showing an XOR feedback path from two tap points.

alternative tap combinations, samples of which are presented in Table 1.

The taps are identical for both XOR-based and XNOR-based LFSRs, although the resulting sequences are different. Multiple alternative tap combinations can also yield maximum-length LFSRs. For example, in the case of a 10-bit LFSR, there are two two-tap combinations that result in a maximal-length sequence ([2,9] and [6,9]), along with 20 four-tap combinations, 28 six-tap combinations, and 10 eight-tap combinations that also satisfy the maximal-length criteria. However, the actual sequence of states varies, depending on the tap selection used.

Program for a maximal-length sequence

A simple program example for a 12-bit XOR-based LFSR using the taps from Table 1 could be as follows:

```
main ()
{
    /* LFSR bit 0   LFSR bit 11 */
    /*          v           v */
    int lfsr = 1; /* Initial value = 0000 0000 0001 */
    int mask = 2369; /* Mask value = 1001 0100 0001 */

    int index; /* Used to store the intermediate result
               /* and index into a pre-calculated table */

    while (1) /* Endlessly loop around generating values */
    {
        index = lfsr * mask; /* Get the values on the taps */
        lfsr >>= 1; /* Shift the LFSR right one bit */
        lfsr |= lookup[index]; /* Insert the input data bit (see notes below) */
        printf("value is %d\n", lfsr); /* Display or otherwise use the output */
        /* Return to the beginning of the loop */
    }
}
```

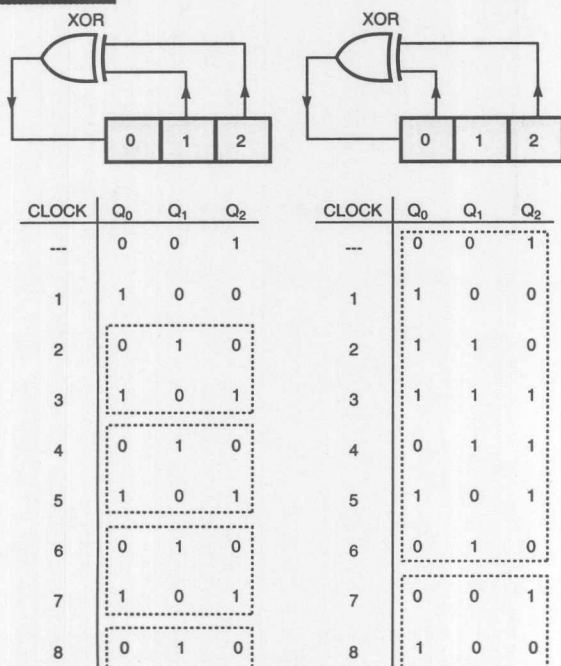
The only tricky point is the line of code above the `printf` statement. After combining the contents of the `lfsr` with the mask in the first line of the `while` loop, the value of `index` contains between 0 and 4 bits set to a 1, and the program must calculate the XOR of these 1s. The quickest way to perform this calculation is to use the `index` variable as an index into a precalculated array of integers (not shown here) called `lookup`. It contains zero values for even numbers of 1s in the `index` and values of 2048 (equivalent to a 1 in the `lfsr`'s bit-0 position) for odd numbers of 1s in the `index`. (Note that the

TABLE 1—TAPS FOR MAXIMAL-LENGTH LFSRS WITH 2 TO 32 BITS

No. of bits	Length of loop	Taps
2	3*	[0,1]
3	7*	[0,2]
4	15	[0,3]
5	31*	[1,4]
6	63	[0,5]
7	127*	[0,6]
8	255	[1,2,3,7]
9	511	[3,8]
10	1023	[2,9]
11	2047	[1,10]
12	4095	[0,3,5,11]
13	8191*	[0,2,3,12]
14	16,383	[0,2,4,13]
15	32,767	[0,14]
16	65,535	[1,2,4,15]
17	131,071*	[2,16]
18	262,143	[6,17]
19	524,287*	[0,1,4,18]
20	1,048,575	[2,19]
21	2,097,151	[1,20]
22	4,194,303	[0,21]
23	8,388,607	[4,22]
24	16,777,215	[0,2,3,23]
25	33,554,431	[2,24]
26	67,108,863	[0,1,5,25]
27	134,217,727	[0,1,4,26]
28	268,435,455	[2,27]
29	536,870,911	[1,28]
30	1,073,741,823	[0,3,5,29]
31	2,147,483,647*	[2,30]
32	4,294,967,295	[1,5,6,31]

*SEQUENCES WHOSE LENGTH IS A PRIME NUMBER.

FIGURE 2



Alternative tap selections generate different state sequences.

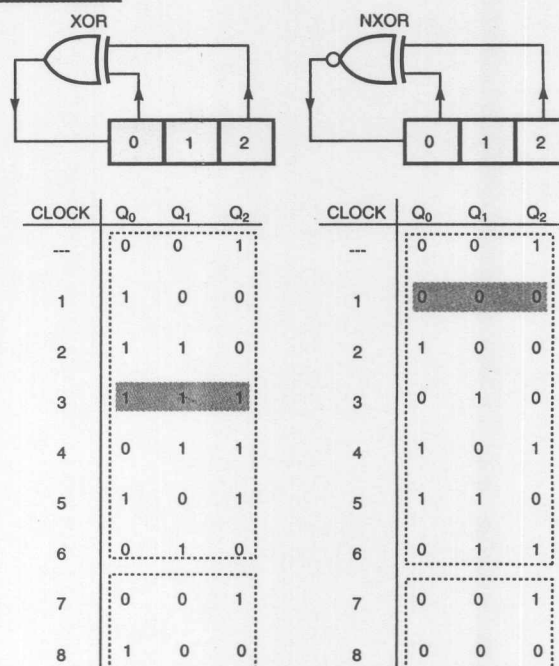
register-bit ordering in this program conforms to the symbols to avoid confusion.)

Consider the case of an 8-bit LFSR, for which the minimum number of taps generating a maximal-length sequence is four. XOR gates come with only two inputs, so a four-input XOR function has to be created using three XOR gates arranged as two levels of logic. Even in those cases where an LFSR does support a minimum of two taps, you may still wish to use more taps (such as eight), which would result in three levels of XOR logic.

One LFSR application where you may wish to use more taps is when you are generating cyclic redundancy checks (CRCs). In this case, the taps are selected such that an error in a single data bit causes the maximum possible disruption to the final contents of the register, referred to as the "checksum value." Thus, in addition to having maximal-length sequences, these LFSRs are also qualified as maximal-displacement.

One problem is that increasing the levels of logic in the combinational feedback path can impact the maximum clocking frequency of the function. A solution to this problem is to

FIGURE 3

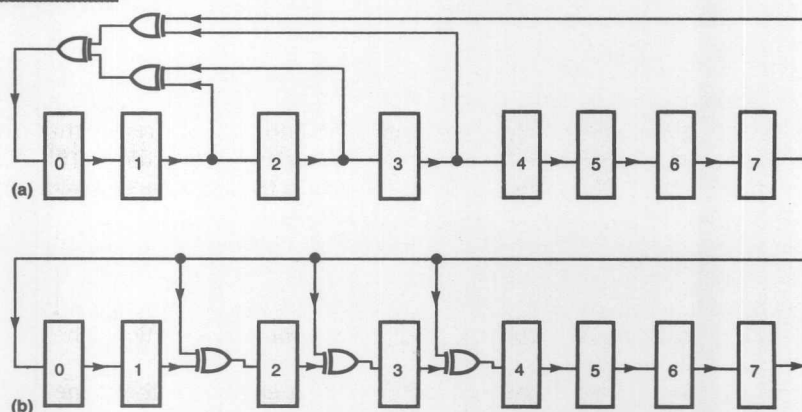


An XOR or an XNOR gate in the feedback path generates different state sequences.

transpose traditional many-to-one implementations into their one-to-many counterparts (Figure 4).

The many-to-one implementation for the 8-bit LFSR has taps at [1,2,3,7]. To convert this implementation into its one-to-many counterpart, you feed the MSB directly into the LSB, and you also individually XOR the MSB with the original taps

FIGURE 4



A many-to-one taps implementation (a) and a one-to-many taps implementation (b) generate different state sequences, but both are maximal-length.

LINEAR-FEEDBACK-SHIFT REGISTERS

(bits 1,2,3 in this example). Although both styles result in maximal-length LFSRs, the actual sequence of values between them differs. Using the one-to-many style means that there is only one level of combinational logic in the feedback path, irrespective of the number of taps being employed.

One of the quirks of an XOR-based LFSR without any form of external data input is when the LFSR contains a value of all zeros. In this case, the shift register continues shifting zeros indefinitely. The same is true for an XNOR-based LFSR with a value of all ones. This problem is of particular concern when you first apply power to the circuit,

because each register bit can randomly initialize with either all zeros or all ones, and the LFSR can, therefore, wake up containing its forbidden value. Therefore, it is necessary to provide a mechanism that initializes the LFSR with a seed value.

One method for loading a seed value is to use registers with reset or set inputs. You can connect a single control signal to the resets on some of the registers and to the sets on others. Thus, when the control signal is in its active state, you load the LFSR with a hard-wired seed value. However, in certain applications, it is desirable to vary the seed value. To achieve

LFSRS HAVE A WEALTH OF APPLICATIONS

Unfortunately, this article can only skim the surface of applications for linear-feedback-shift registers (LFSRs). LFSRs are used in a wealth of other applications, some examples of which follow:

Encryption and decryption: You can use the pseudorandom sequence of values generated by an LFSR in the encryption (scrambling) and decryption (unscrambling) of data in communication systems. In the case of digital data, you can simply XOR the data stream with the output of an LFSR to generate an encrypted signal. (A similar process at the receiver decrypts the incoming signal.)

Data communications: For certain communication applications in environments with significant background noise, modulating the data using a maximal-length LFSR provides the modulated signal an auto-correlation function that lets you recover the data despite a S/N ratio of many decibels.

Data integrity: A traditional application for LFSRs is in cyclic-redundancy-check (CRC) calculations. CRC calculators let you detect errors in data communications by using the stream of transmitted data bits to modify the values fed back to the LFSR. The final CRC value stored in the LFSR, which is known as the checksum, depends on every bit in the data stream. The receiver compares an internally generated checksum with the checksum sent by the transmitter to determine whether any corruption occurred during transmission. This form of error detection is efficient because of the small number of bits that have to be transmitted in addition to the data.

You can use similar techniques to

generate a CRC based on a file's contents. The resulting checksum is attached to that file and can subsequently be used to make sure the contents of the file have not been corrupted. An offshoot of this application is used in the detection of computer viruses, although more sophisticated viruses have evolved that can counteract these measures.

Fault test: Another application for CRC calculators is in the circuit-board test strategy known as "functional test." You plug the board into a functional tester, and the tester applies a pattern of signals to the board's inputs. After allowing sufficient time for propagation effects to settle, the tester compares the actual values observed on the outputs with a set of expected values that are stored in the system. This process is repeated for a large number of input sequences.

If the board fails the preliminary tests, you can employ a more sophisticated form of analysis, called a "guided probe," to identify the cause of the failure. The tester instructs the operator to place the probe at a location on the board and then to rerun the entire sequence of test patterns. The tester compares the actual sequence of values the probe sees with a sequence of expected values stored in the system. You repeat this process until the tester locates the faulty component or track.

A major consideration when using guided-probe testing is the amount of data you expect to store. One way to minimize the expected data is to use LFSR-based CRC calculators. You can pass the sequence of expected values

through a CRC calculator implemented in software. You can also pass the sequence of actual values seen by the guided probe through an identical CRC calculator implemented in hardware. The calculated checksum values are also known as "signatures," and the guided probe test based on this technique is known as "signature analysis." Irrespective of the number of test patterns used, the system has to store only a single signature for each track. In addition, for each application, the tester has to compare only the signature gathered by the probe with the expected signature stored in the system. Therefore, the compressed data result in storage requirements that are much smaller and comparison times that are much shorter than the uncompressed approach.

Built-in self-test: Another test strategy you may want to employ is built-in self-test (BIST). Devices using BIST contain special test generators and result-gathering circuits, both of which you can implement using LFSRs.

Pseudorandom numbers: Many computer programs, including games, digital and analog simulators, and graphic packages, rely on random sequences. You can generate pseudorandom sequences using LFSRs. In fact, pseudorandom numbers have an advantage over truly random numbers, because many computer applications typically require repeatability. However, designers also need the ability to modify the seed value of the pseudorandom-number generator, to create alternative pseudorandom sequences.

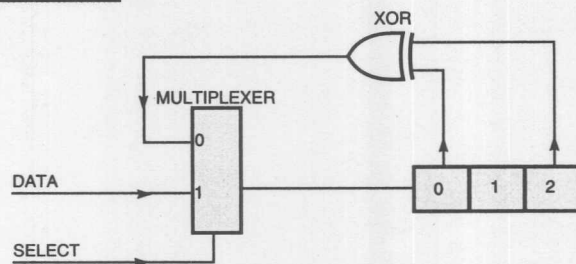
LINEAR-FEEDBACK-SHIFT REGISTERS

this variation, you can include a multiplexer at the LFSR's input (Figure 5). When you select the data input, the device functions as a standard-shift register, and you can load any seed value. After loading the seed value, select the feedback path, and the device returns to its LFSR mode of operation.

Use of LFSRs in FIFO memory

The fact that an LFSR generates a pseudorandom sequence of values is irrelevant in some applications. Consider an SRAM-based, 4-bit $\times$ 16 word FIFO memory device. The write and read pointers are essentially 4-bit registers whose outputs are processed by 4-to-16 decoders to select one of the 16 words in the memory array. A *reset* input initializes the device, primarily by clearing the write and read pointers such that they both point to the same memory word. The initialization also places the *empty* output in its active state

FIGURE 5



A multiplexer at the LFSR's input lets you vary the register's seed value.

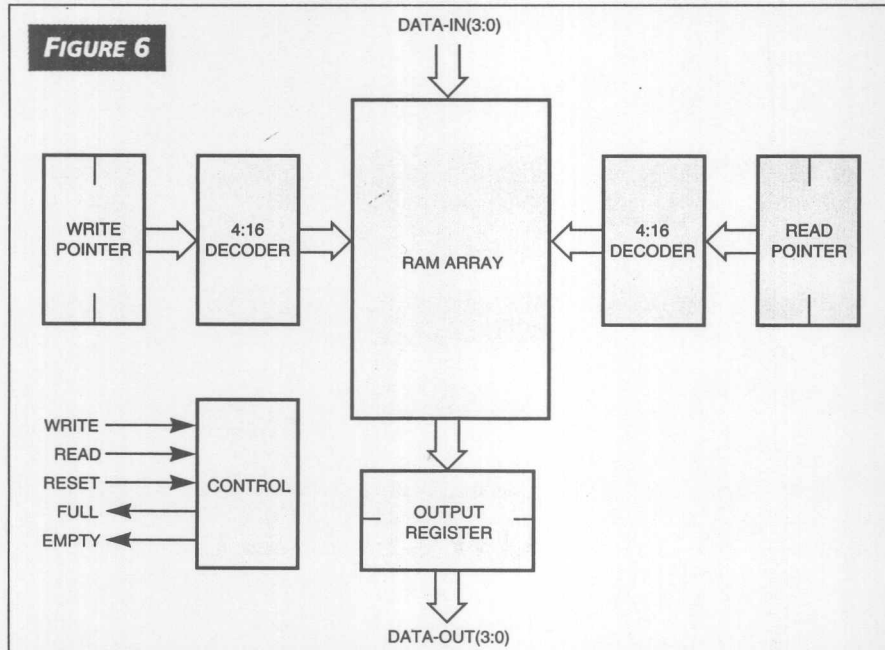
and places the *full* output in its inactive state.

The write and read pointers chase each other around the memory array in an endless loop. An active edge on the *write* input causes any data on the data-in bus to be written into the word indicated by the write pointer. The *empty* output goes into its inactive state, and the write pointer increments to point to the next empty word. You can write data into the FIFO until all the words in the array contain data. When the write pointer catches up to the read pointer, the *full* output goes into its active state, and no more data can be written into the device.

An active edge on the *read* input causes the data pointed to by the read pointer to be copied into the output register. The *full* output goes into its inactive state, and the read pointer increments to point to the next word that contains data. You can read data out of the FIFO until the array is empty. When the read pointer catches up to the write pointer, the *empty* output goes into its active state, indicating that you can't read any more data out of the device.

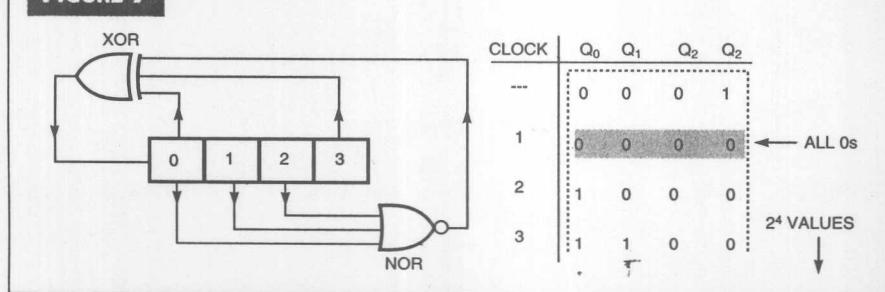
The write and read pointers for a 16-word FIFO are often implemented using 4-bit binary counters. However, a moment's reflection reveals that there is no intrinsic advantage to a binary sequence for this application. The sequence generated by a 4-bit LFSR could serve equally well. In addition, although the combinational "next-state" logic for the 4-bit binary counter requires a number of AND and OR

FIGURE 6



A block diagram of a FIFO memory shows the write and read pointers along with their decoders.

FIGURE 7



The addition of a NOR gate in the feedback path modifies the state sequence to 2ⁿ values.

LINEAR-FEEDBACK-SHIFT REGISTERS

gates, the combinational logic for the 4-bit LFSR consists only of a single two-input XOR gate. This situation means that the LFSR requires fewer tracks and is significantly more efficient in silicon real estate. In addition, the LFSR's feedback passes only through a single level of logic, although the binary counter's feedback passes through multiple levels of logic. This fact means that the new data value is available for the LFSR sooner, and, therefore, you can use a higher clock frequency. The differentiations become more pronounced for FIFOs with more words requiring pointers with more bits.

The downside to the 4-bit LFSRs in this FIFO example is that, unlike a binary counter's sequence of 16 values (2^4), the LFSR sequences only through 15 values (2^4-1). If you require an LFSR to sequence through every possible value, there is a simple solution (Figure 7). For the value when all the bits are zero, the preceding value must have comprised a logic 1 in the MSB, and logic 0s in the remaining bits. In an unmodified LFSR, the next clock would result in a logic 1 in the LSB and logic 0s in the remaining bits. However, in the modified LFSR, the output from the NOR gate is a logic 0 for every case, except when the inputs to the NOR gate are all zero. This condition forces the NOR gate's output to a logic 1, which inverts the usual output from the XOR gate. This condition, in turn, causes the sequence to enter the all-0 value first and then resume its normal sequence. In the case of LFSRs with XNOR feedback paths, the NOR gate can be replaced with an AND gate, which causes the sequence to cycle through the value where all of the bits are one.

In some applications, you may need to make use of an LFSR's previous value. For example, in certain FIFO implementations, when the write pointer is set to the location preceding the one pointed to by the read pointer, this situation indicates a full condition. This situation implies that you must use a comparator to compare the current value in the

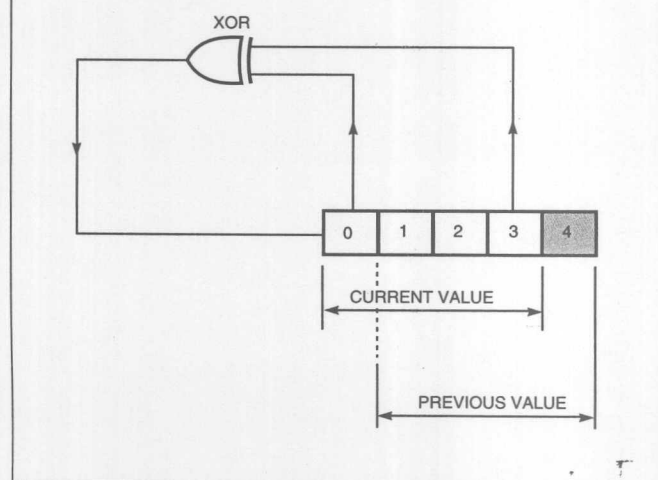
write pointer with the previous value in the read-pointer. Similarly, when the read pointer is set to the location preceding the one pointed to by the write pointer, this situation indicates an empty condition. This condition implies that you must use a second comparator to compare the current value in the read pointer with the previous value in the write pointer.

In the case of binary counters, you can use two techniques to access the previous value in the sequence. The first approach requires a set of "shadow registers." Every time the clock increments the counter, the current contents are first copied into the shadow registers. Alternatively, you can use a block of combinational logic to decode the previous value from the current value. Both of these techniques involve substantial overhead for the additional logic. In contrast, LFSRs inherently remember their previous value. All you need is the addition of a single register bit appended to the MSB position (Figure 8).

LFSRs are simple to construct and are useful for a wide variety of applications. However, choosing the optimal polynomial, or tap points, for a particular application is a task that is usually reserved for a master of the mystic arts, and the math can be hairy enough to make a grown man cry. Much like the Ouroboros, LFSRs keep on rearing their heads again and again and again, because however much you know, there are always more tricks to learn. The great psychoanalyst Carl Jung speculated that the Ouroboros could be part of the collective consciousness, so had he dabbled in electronics, he may well have considered LFSRs to form part of the digital consciousness. Hold onto that couch, Carl; I'm on my way!

EDN

FIGURE 8



You can access the previous state value from an LFSR by adding one more register stage.

Author's biography

Clive "Max" Maxfield, member of the technical staff (MTS) at Intergraph Electronics (Huntsville, AL), helps specify Intergraph's electronic-design-automation (EDA) products (phone (800) 837-4237). In addition to numerous technical articles and papers, Maxfield is also the author of *Bebop to the Boolean Boogie: An Unconventional Guide to Electronics* (ISBN 1-878707-22-1). To order, phone (800) 247-6553.

VOTE

Please use the Information Retrieval Service card to rate this article (circle one):

High Interest
589

Medium Interest
599

Low Interest
600